

The diagram illustrates a Petri net with two main components, P1 and P2, enclosed in dashed rectangular boxes. P1 is on the left and P2 is on the right. A solid horizontal arrow points from P1 to P2, representing a communication channel. In P1, there is a vertical sequence of three circular places. The top place has an incoming wavy line from above. The middle place has an outgoing wavy line to the top place. The bottom place has an outgoing wavy line to the middle place. A transition labeled 'PUT' is located to the right of the bottom place in P1. A dashed line connects the 'PUT' transition to the input place of the channel in P2. In P2, there is a vertical sequence of three circular places. The top place has an incoming wavy line from above. The middle place has an outgoing wavy line to the top place. The bottom place has an outgoing wavy line to the middle place. A transition labeled 'GET' is located to the right of the bottom place in P2. A dashed line connects the 'GET' transition to the output place of the channel in P1. The channel is represented by two thick vertical bars, one in P1 and one in P2, connected by a solid horizontal arrow. The input place of the channel in P2 is the top place of the vertical sequence in P2, and the output place of the channel in P1 is the bottom place of the vertical sequence in P1.

Figure 1

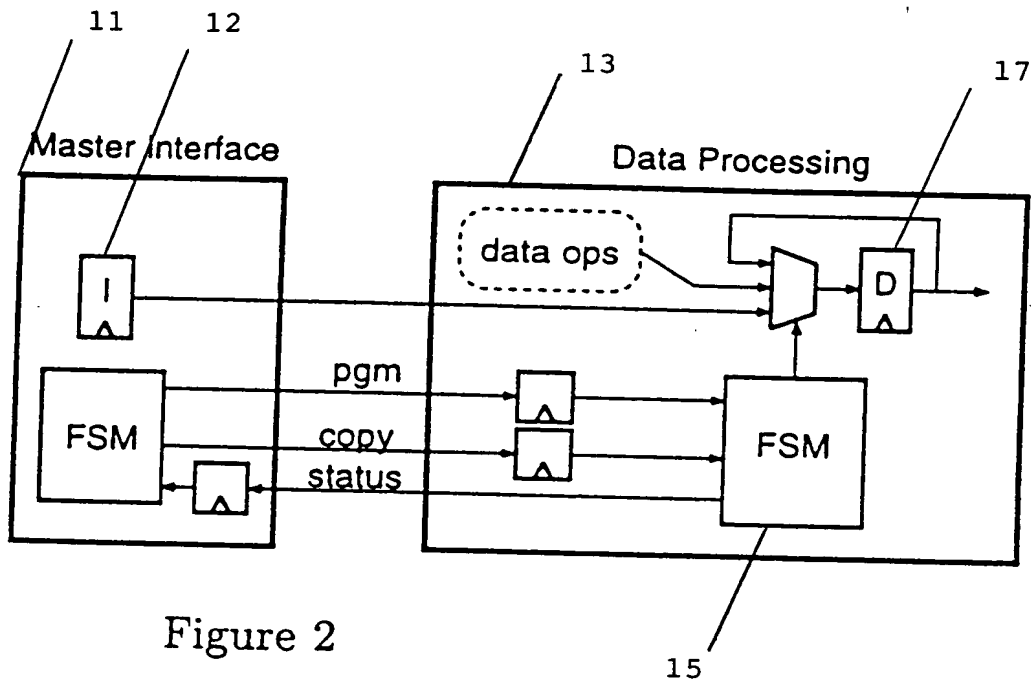


Figure 2

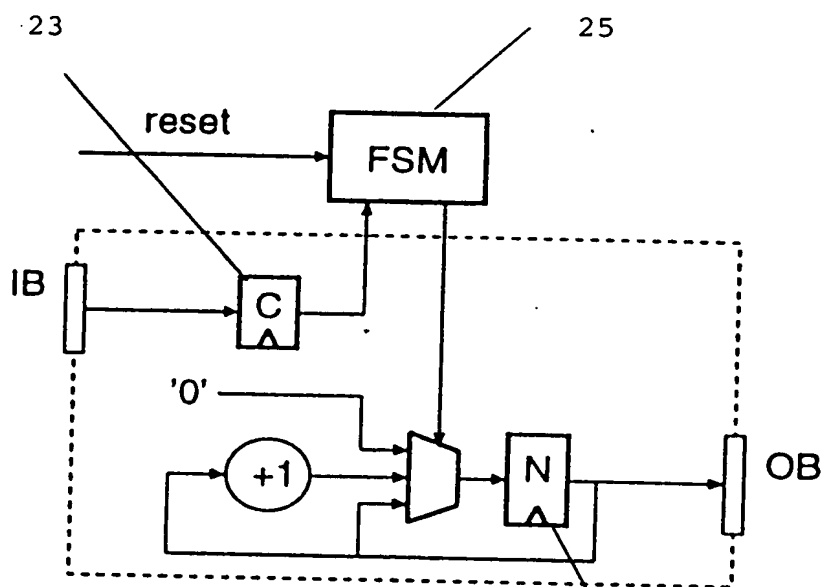


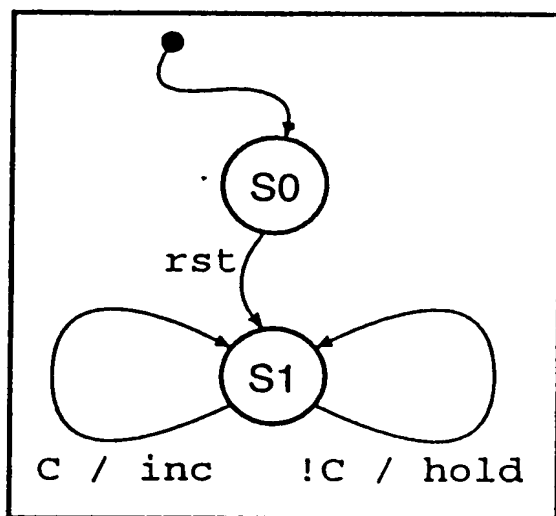
Figure 3

21

669720" 6896450

6667E0"666E2660

FSM



```
register C;  
register N;  
input    IB;  
output   OB;
```

```
rst:    N = 0;  
        C = IB;
```

```
inc:    N = N + 1;  
        C = IB;  
        OB = N;
```

```
hold:   C = IB;  
        OB = N;
```

Figure 4

Figure 5 illustrates a synchronization mechanism between two processes, P1 and P2, using a synchronizer object.

The diagram is divided into two parts, 1 and 2, showing the state of the system at different points in time.

**Part 1:** Process P1 is shown with a sequence of operations: PUT, ack/read, req0, !ack/read, req1, p\_req, and p\_ack. Process P2 is shown with a hook function. A synchronizer object is shown with an expand function. The hook function is called by both processes. The expand function is called by the synchronizer object.

**Part 2:** Process P1 is shown with a sequence of operations: PUT, ack/read, req0, !ack/read, req1, p\_req, and p\_ack. Process P2 is shown with a hook function. A synchronizer object is shown with an expand function. The hook function is called by both processes. The expand function is called by the synchronizer object.

The diagram shows the flow of control and data between the processes and the synchronizer object. The hook function is used to coordinate the execution of the PUT operation. The expand function is used to expand the scope of the synchronization. The sequence of operations (PUT, ack/read, req0, !ack/read, req1, p\_req, p\_ack) represents the execution of the PUT operation and the subsequent communication between the processes and the synchronizer object.

Figure 5



The diagram illustrates the program state transition logic. It features a state transition graph with three main states: **s1**, **s2**, and a central state (represented by a circle). Transitions are labeled with conditions and actions:

- From **s1** to **s2**: `prog & !copy / read, status1`
- From **s2** to **s1**: `prog & copy / read, status1, upd`
- From the central state to **s1**: `!prog / status0`
- From the central state to **s2**: `prog / read, status0`
- From **s1** to the central state: `read, status0`
- From **s2** to the central state: `read, status0`

A **prog\_itf object** is shown with two methods: `::hook()` and `::expand()`. A stack **D** is also depicted. The diagram is annotated with numbers 41, 43, 45, and 47.

Figure 7

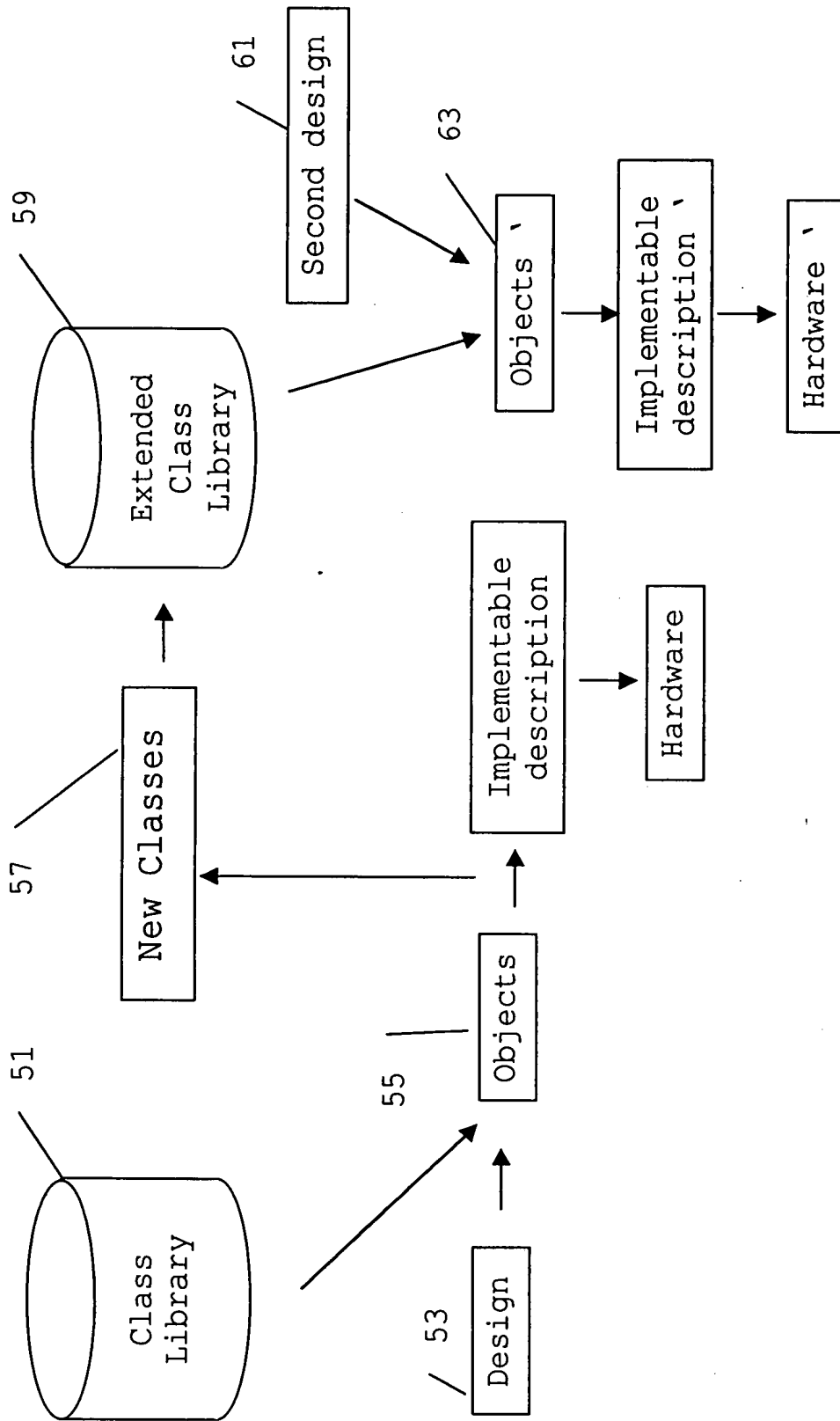


Figure 8